软件构造第1次实验

实验报告

**班级**：软件工程14-3

**学号**：201401061038

**姓名**：张正锟

2016年3月30日

目录

[一、 设计 3](#_Toc447119762)

[1. 飞机类的继承 3](#_Toc447119763)

[2. 单例模式的应用 4](#_Toc447119764)

[3. 飞机的各种行为（策略模式） 5](#_Toc447119765)

[4. 观察者模式的应用 6](#_Toc447119766)

[5. 未使用工厂方法模式所遇到的问题 7](#_Toc447119767)

[6. 使用工厂方法模式 7](#_Toc447119768)

[7. 使用装饰者模式 8](#_Toc447119769)

[二、 实现 8](#_Toc447119770)

[1. 基础设施：表示飞机位置的Position类 8](#_Toc447119771)

[2. 统计我机分数的ScoreCounter单例类 9](#_Toc447119772)

[3. 飞机的颜色行为实现 10](#_Toc447119773)

[4. 飞机飞行行为的实现 11](#_Toc447119774)

[5. 飞机发声行为的实现 11](#_Toc447119775)

[6. 敌机打掉后观察者的实现 12](#_Toc447119776)

[7. 飞机工厂实现 12](#_Toc447119777)

[8. 飞机装饰者实现 13](#_Toc447119778)

[9. 飞机基类（抽象）的实现 14](#_Toc447119779)

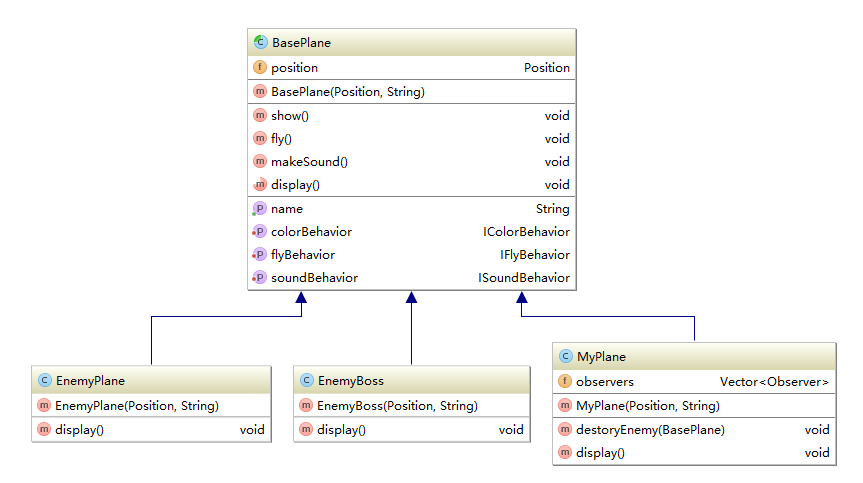
[10. 我机的实现 15](#_Toc447119780)

[11. 敌机战机的实现 16](#_Toc447119781)

[12. 敌机Boss的实现 16](#_Toc447119782)

# 设计

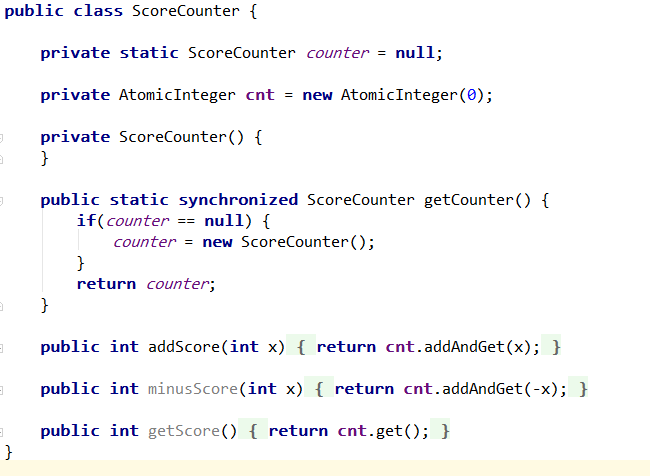
## 飞机类的继承



战机，敌机都是飞机：应用继承的思想

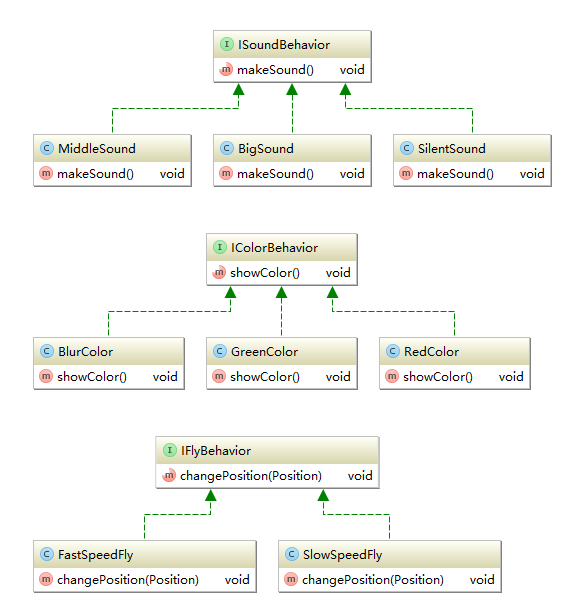
定义飞机抽象类，具有飞行、展示颜色、发出轰鸣的行为

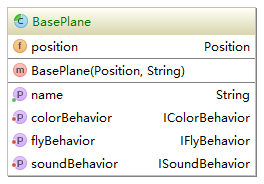
## 单例模式的应用

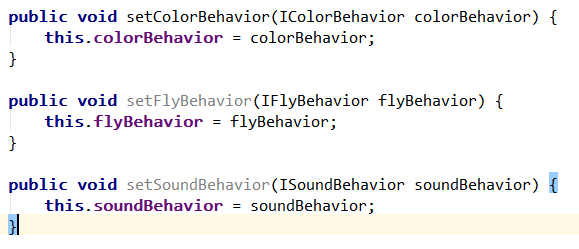


单例模式保证“分数”在游戏中只有一个实例类

## 飞机的各种行为（策略模式）

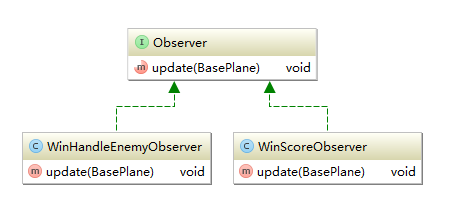


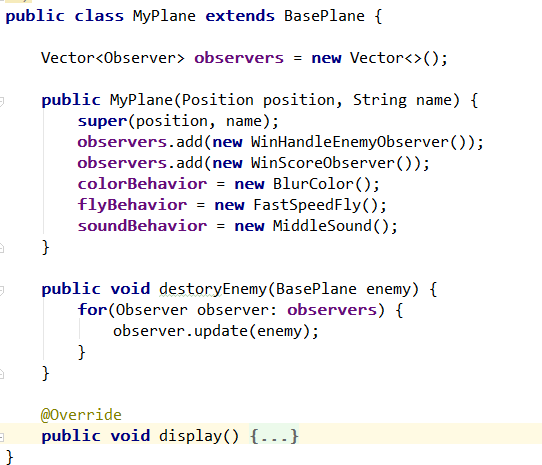




不同的飞机有不同的特点，应用策略模式将飞机行为和飞机分离

## 观察者模式的应用

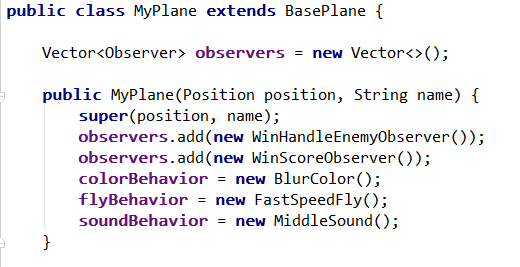




观察者模式实现当敌机被打掉，通知各个Observer实现加分以及去掉敌机图片

内部含有一个Vector，存放当敌机被打掉时所通知的Observer

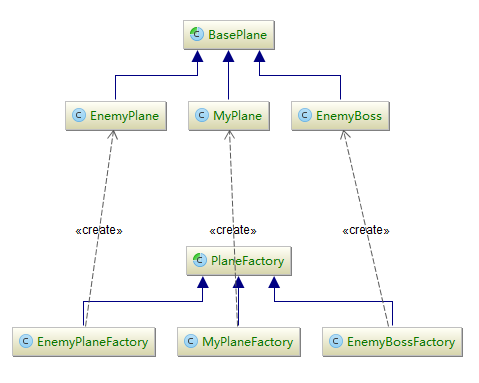
## 未使用工厂方法模式所遇到的问题



可以看到，为了组装飞机还需要另外实例化飞机的颜色、飞行、发声行为，而这些飞机的行为是与调用者无关的，耦合度太高。并且非常不利于扩展。

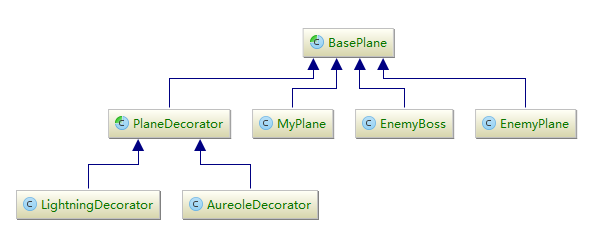
另外，本例中各种行为还是比较具体的，在实际应用中，可能这些产品的组件也都是抽象的，调用者根本不知道怎样组装产品。假如使用工厂方法的话，整个架构就显得清晰了许多。

## 使用工厂方法模式



使用工厂方法后，调用端的耦合度大大降低了。并且对于工厂来说，是可以扩展的，以后如果想组装其他的飞机，只需要再增加一个工厂类的实现就可以。无论是灵活性还是稳定性都得到了极大的提高。

## 使用装饰者模式



装饰对象可以方便增加一些附加功能。这样就确保了在运行时，不用修改给定对象的结构就可以在外部增加附加的功能。在面向对象的设计中，通常是通过继承来实现对给定类的功能扩展。

# 实现

## 基础设施：表示飞机位置的Position类

**public class** Position {  
 **int x**;  
 **int y**;  
  
 **public** Position() {  
 **x** = **y** = 0;  
 }  
  
 **public** Position(**int** x, **int** y) {  
 **this**.**x** = x;  
 **this**.**y** = y;  
 }  
  
 **public int** getX() {  
 **return x**;  
 }  
  
 **public void** setX(**int** x) {  
 **this**.**x** = x;  
 }  
  
 **public int** getY() {  
 **return y**;  
 }  
  
 **public void** setY(**int** y) {  
 **this**.**y** = y;  
 }  
  
 **public void** move(**int** dx, **int** dy) {  
 **x** += dx;  
 **y** += dy;  
 }  
  
 **public static** Position random() {  
 Random random = **new** Random();  
 **return new** Position(random.nextInt(100), random.nextInt(100));  
 }  
  
 @Override  
 **public** String toString() {  
 **return "("** + **x** + **", "** + **y** + **")"**;  
 }  
}

## 统计我机分数的ScoreCounter单例类

**public class** ScoreCounter {  
  
 **private static** ScoreCounter *counter* = **null**;  
  
 **private** AtomicInteger **cnt** = **new** AtomicInteger(0);  
  
 **private** ScoreCounter() {  
 }  
  
 **public static synchronized** ScoreCounter getCounter() {  
 **if**(*counter* == **null**) {  
 *counter* = **new** ScoreCounter();  
 }  
 **return** *counter*;  
 }  
  
 **public int** addScore(**int** x) {  
 **return cnt**.addAndGet(x);  
 }  
  
 **public int** minusScore(**int** x) {  
 **return cnt**.addAndGet(-x);  
 }  
  
 **public int** getScore() {  
 **return cnt**.get();  
 }  
}

## 飞机的颜色行为实现

**public interface** IColorBehavior {  
 */\*\*  
 \* 展示飞机颜色方法  
 \*/* **void** showColor();  
}

**public class** BlurColor **implements** IColorBehavior {  
 @Override  
 **public void** showColor() {  
 System.***out***.println(**"飞机颜色为蓝色..."**);  
 }  
}

**public class** GreenColor **implements** IColorBehavior {  
 @Override  
 **public void** showColor() {  
 System.***out***.println(**"飞机颜色为绿色..."**);  
 }  
}

**public class** RedColor **implements** IColorBehavior {  
 @Override  
 **public void** showColor() {  
 System.***out***.println(**"飞机颜色为红色..."**);  
 }  
}

## 飞机飞行行为的实现

**public interface** IFlyBehavior {  
 */\*\*  
 \* 改变飞机位置  
 \** ***@param pos*** *飞机原位置  
 \*/* **void** changePosition(Position pos);  
}

**public class** FastSpeedFly **implements** IFlyBehavior {  
 @Override  
 **public void** changePosition(Position pos) {  
 pos.move(0, 10);  
 System.***out***.println(**"飞机向前移动了10个单位，目前坐标："** + pos);  
 }  
}

**public class** SlowSpeedFly **implements** IFlyBehavior {  
 @Override  
 **public void** changePosition(Position pos) {  
 pos.move(0, 3);  
 System.***out***.println(**"飞机向前移动了3个单位，目前坐标："** + pos);  
 }  
}

## 飞机发声行为的实现

**public interface** ISoundBehavior {  
 */\*\*  
 \* 发出声音的方法  
 \*/* **void** makeSound();  
}

**public class** BigSound **implements** ISoundBehavior {  
 @Override  
 **public void** makeSound() {  
 System.***out***.println(**"飞机发出200分贝巨大轰鸣..."**);  
 }  
}

**public class** MiddleSound **implements** ISoundBehavior {  
 @Override  
 **public void** makeSound() {  
 System.***out***.println(**"飞机发出100分贝轰鸣..."**);  
 }  
}

**public class** SilentSound **implements** ISoundBehavior {  
 @Override  
 **public void** makeSound() {  
 System.***out***.println(**"无声飞机...静音..."**);  
 }  
}

## 敌机打掉后观察者的实现

**public interface** Observer {  
 **void** update(BasePlane enemy);  
}

**public class** WinHandleEnemyObserver **implements** Observer {  
 @Override  
 **public void** update(BasePlane enemy) {  
 System.***out***.println(**"敌机"** + enemy.getName() + **"已经阵亡~"**);  
 }  
}

**public class** WinScoreObserver **implements** Observer {  
  
 @Override  
 **public void** update(BasePlane enemy) {  
 ScoreCounter counter = ScoreCounter.*getCounter*();  
 **if**(enemy **instanceof** EnemyPlane) {  
 counter.addScore(7);  
 System.***out***.println(**"帅呆了~，击中敌机增加7分~"**);  
 } **else if**(enemy **instanceof** EnemyBoss) {  
 counter.addScore(77);  
 System.***out***.println(**"NB啊~，击中大Boss增加77分~"**);  
 }  
 }  
}

## 飞机工厂实现

**public abstract class** PlaneFactory {  
 **abstract public** BasePlane createPlane(String name);  
}

**public class** MyPlaneFactory **extends** PlaneFactory {  
 @Override  
 **public** BasePlane createPlane(String name) {  
 BasePlane res = **new** MyPlane(Position.*random*(), name);  
 res.setColorBehavior(**new** BlurColor());  
 res.setFlyBehavior(**new** FastSpeedFly());  
 res.setSoundBehavior(**new** MiddleSound());  
 **return** res;  
 }  
}

**public class** EnemyPlaneFactory **extends** PlaneFactory {  
 @Override  
 **public** BasePlane createPlane(String name) {  
 BasePlane res = **new** EnemyPlane(Position.*random*(), name);  
 res.setColorBehavior(**new** GreenColor());  
 res.setFlyBehavior(**new** SlowSpeedFly());  
 res.setSoundBehavior(**new** SilentSound());  
 **return** res;  
 }  
}

**public class** EnemyBossFactory **extends** PlaneFactory {  
 @Override  
 **public** BasePlane createPlane(String name) {  
 BasePlane res = **new** EnemyBoss(Position.*random*(), name);  
 res.setColorBehavior(**new** RedColor());  
 res.setFlyBehavior(**new** FastSpeedFly());  
 res.setSoundBehavior(**new** BigSound());  
 **return** res;  
 }  
}

## 飞机装饰者实现

**public abstract class** PlaneDecorator **extends** BasePlane {  
 **public** PlaneDecorator(Position position, String name) {  
 **super**(position, name);  
 }  
}

*/\*\*  
 \* 装饰者：光环  
 \* Created by kun on 2016/3/22.  
 \*/***public class** AureoleDecorator **extends** PlaneDecorator {  
  
 BasePlane **plane**;

**public** AureoleDecorator(Position position, String name) {  
 **super**(position, name);  
 }  
  
 @Override  
 **public void** display() {  
 **plane**.display();  
 System.***out***.println(**"带光环~"**);  
 }  
}

*/\*\*  
 \* 装饰者：闪电  
 \* Created by kun on 2016/3/22.  
 \*/***public class** LightningDecorator **extends** PlaneDecorator {  
  
 BasePlane **plane**;  
  
 **public** LightningDecorator(Position position, String name) {  
 **super**(position, name);  
 }  
  
 @Override  
 **public void** display() {  
 **plane**.display();  
 System.***out***.println(**"喷气~~~"**);  
 }  
}

## 飞机基类（抽象）的实现

**public abstract class** BasePlane {  
  
 **protected** String **name**;  
 **protected** Position **position**;  
 **protected** IColorBehavior **colorBehavior**;  
 **protected** IFlyBehavior **flyBehavior**;  
 **protected** ISoundBehavior **soundBehavior**;  
  
 **public** BasePlane(Position position, String name) {  
 **this**.**position** = position;  
 **this**.**name** = name;  
 }  
  
 **public** String getName() {  
 **return name**;  
 }  
  
 **public void** setColorBehavior(IColorBehavior colorBehavior) {  
 **this**.**colorBehavior** = colorBehavior;  
 }  
  
 **public void** setFlyBehavior(IFlyBehavior flyBehavior) {  
 **this**.**flyBehavior** = flyBehavior;  
 }  
  
 **public void** setSoundBehavior(ISoundBehavior soundBehavior) {  
 **this**.**soundBehavior** = soundBehavior;  
 }  
  
 **public void** show() {  
 **colorBehavior**.showColor();  
 }  
  
 **public void** fly() {  
 **flyBehavior**.changePosition(**position**);  
 }  
  
 **public void** makeSound() {  
 **soundBehavior**.makeSound();  
 }  
  
 **public abstract void** display();  
}

## 我机的实现

**public class** MyPlane **extends** BasePlane {  
  
 Vector<Observer> **observers** = **new** Vector<>();  
  
 **public** MyPlane(Position position, String name) {  
 **super**(position, name);  
 **observers**.add(**new** WinHandleEnemyObserver());  
 **observers**.add(**new** WinScoreObserver());  
 }  
  
 **public void** destoryEnemy(BasePlane enemy) {  
 **for**(Observer observer: **observers**) {  
 observer.update(enemy);  
 }  
 }  
  
 @Override  
 **public void** display() {  
 System.***out***.print(**"我就是"** + **name**);  
 **colorBehavior**.showColor();  
 }  
}

## 敌机战机的实现

**public class** EnemyPlane **extends** BasePlane {  
  
 **public** EnemyPlane(Position position, String name) {  
 **super**(position, name);  
 }  
  
 @Override  
 **public void** display() {  
 System.***out***.print(**"我是普通敌机"** + **name**);  
 **colorBehavior**.showColor();  
 }  
}

## 敌机Boss的实现

**public class** EnemyBoss **extends** BasePlane {  
  
 **public** EnemyBoss(Position position, String name) {  
 **super**(position, name);  
 }  
  
 @Override  
 **public void** display() {  
 System.***out***.print(**"我是大Boss"** + **name**);  
 **colorBehavior**.showColor();  
 }  
}